Runtime for Application Resiliency

Larry Kaplan, Cray Inc. (lkaplan@cray.com)

Current trends in high performance computing (HPC) system design are rapidly increasing the number of transistors and structures used in the nodes of such systems. While socket counts are also expected to grow somewhat, it is the increase in complexity on the node that is expected to grow even more with respect to current systems. In addition, in order to achieve that increase in local complexity while staying within both physical and power constraints, future IC technology is expected to be very dense and to run with very low voltages, resulting in the possibility of significantly increased failure rates, especially for large systems (Kogge, 2008). These increases are likely to render traditional strategies of external I/O based checkpoint restart (CPR) as unworkable for application resiliency (Schroeder & Gibson, 2007). In fact, while CPR to local storage may extend the life of such a globally coordinated resilience strategy, the global coordination itself may struggle in the face of increasing failure rates, if not essentially continuous failures. Any failure that occurs while another is being handled can potentially eliminate the possibility of forward progress for globally coordinated resiliency schemes.

While application resiliency is clearly a crosscutting topic that touches upon many aspects of system design, significant work is required just in the areas of operating systems (OS) and runtime. By runtime we mean the software that is directly linked to the application. This paper proposes research to address issues of application resiliency at the runtime level. Some operating system support and infrastructure will also be required.

Description of Research Direction

Some of the work required for application resiliency lies in the program model infrastructure. While compiler work is out-of-scope for this particular discussion, there are several runtime infrastructure issues that need to be addressed before a full-blown application resiliency solution can be considered.

At the time of writing this document, MPI-3 FT (MPI 3.0 Fault Tolerance Working Group, 2009) appears to be headed towards providing the required infrastructure. This comes down to the ability for an MPI application to continue in the face of a node failure. In addition, MPI resilience to network failures has largely been solved with reasonable overheads. There are some other pieces of required OS infrastructure that also must be provided, such as a resilient process management interface (PMI) and job launch and control system, but those efforts also appear to be progressing sufficiently.

The situation is quite different for global address space (GAS) languages such as the partitioned GAS languages, Unified Parallel C (UPC) and Fortran with Co-arrays (CAF). Newer GAS languages such as Chapel also have similar concerns. These languages tend to have finer grained communication than MPI and so are more susceptible to overheads in protecting their communications. In addition, these languages often support atomic memory operations (AMOs). Making AMOs efficiently resilient to network failures is even a bigger task, likely requiring some help from the hardware. For node failures, some of the interface and semantic issues will necessarily appear at the compiler and language level and so are outside of the scope of this discussion. However good runtime support will also be required to support the compiler provided semantics.

Finally, in order to handle efficient and independent roll back and recovery, an explicit notion of the input and output dependencies for the various pieces of work that can make up an application will be essential for understanding and strategizing the runtime recovery from various failures. This will require some form of representation in the runtime.

Once the above infrastructure is available, then further progress is possible on the implementation of uncoordinated rollback and recovery at a fairly fine-grained level. Ultimately it is highly desirable that the application resiliency

solutions allow for independent recovery so as to allow multiple failures to be handled simultaneously. One current line of research of this type is that of Containment Domains (Sullivan, Yoon, & Erez, 2011). There are several other related techniques that could also be investigated. Part of the additional work required to pursue such strategies is the definition of runtime failure reporting and delivery mechanisms, both directly to the programmer, where explicit failure management strategies are used, and to the runtime itself when additional automation is present.

Challenges Addressed

Network Failure Handling: Network failures in non-MPI programs are not well addressed in current systems and will need to be if such programming models are expected to be useful in exascale systems.

Node Failure Handling: Similar to network failures, node failures are also not well addressed in non-MPI programs in current systems. We expect that the MPI-3 standards body will successfully address this for MPI programs, but little progress has been made for other models.

Scalability: By enabling the ability to handle failures in as local a manner as the application allows (often quite local), applications will be better able to scale in the face of potentially increasing failure rates.

Productivity: Having the runtime manage these issues as automatically as possible improves the ability of the programmer to focus on the science or other goals of their program. It is likely that most of the network failure handling can be hidden from the programmer, caveat the possibility that some resilient operations will have higher overhead than equivalent non-resilient ones. Full automation for node failures is less likely in the near future, but by providing some of the described runtime support, more of the burden of understanding failure implications can be assumed by the runtime rather than requiring programmer attention. A near term goal is to allow the programmer to think locally about individual routine resiliency, rather than having to always consider the entire application as is commonplace today.

Maturity

Some work has begun in these areas, especially in the area of resilience to the various failures for MPI. So there is clearly interest in this work and an expectation that it will be useful. This proposal builds on the work for MPI, and also attempts to bring the concepts to other programming models.

Uniqueness

This kind of resiliency for tightly coupled, large, parallel applications is rather unique to HPC. The issues do not become severe until very large systems are considered.

Novelty

For MPI, this work would be one of a very few that are attempting very fine-grained rollback and recovery, potentially down to the level of an individual function or portion of a computation kernel. The author is unaware of much activity in this space outside of MPI, except potentially in ARMCI (Vishnu, Van Dam, De Jong, Balaji, & Song, 2010).

Applicability

Given the necessary conditions of application failure rates in tightly coupled applications to justify this work, it is not clear if there are other areas of computing where this work would be applicable. Certain high failure rate environments may be relevant, especially where some non-trivial amount of parallelism is required.

Effort

This work consists of infrastructure provisioning and research into the actual application resiliency techniques. Combining these efforts would require approximately 4 FTEs for several years.

Works Cited

Kogge, P. e. (2008). *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. DARPA IPTO.

MPI 3.0 Fault Tolerance Working Group. (2009). *MPI 3.0 Fault Tolerance*. Retrieved 2012, from http://meetings.mpi-forum.org/mpi3.0_ft.php

Schroeder, B., & Gibson, G. A. (2007). Understanding Failures in Petascale Computers. *Jounal of Physics: Conference Series 78* .

Sullivan, M., Yoon, D. H., & Erez, M. (2011). *Containment Domains: A Full-System Approach to Computational Resiliency*. The University of Texas at Austin, Department of Electrical and Computer Engineering. Austin: The University of Texas at Austin.

Vishnu, A., Van Dam, H., De Jong, W., Balaji, P., & Song, S. (2010). Fault-tolerant communication runtime support for data-centric programming models. *HiPC 2010*. IEEE.